**Česká zemědělská univerzita v Praze**

**Provozně ekonomická fakulta**

**Katedra informačního inženýrství**


**Bakalářská práce**

# Analýza dopravních nehod na území České republiky

**Artem Shchelkunov**

# ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Artem Shchelkunov

Systémové inženýrství a informatika
Informatika

Název práce

**Analýza dopravních nehod na území České republiky**

Název anglicky

**Analysis of traffic accidents in the Czech Republic**

---

**Cíle práce**

Tato práce se zaměřuje na analýzu údajů o dopravních nehodách v České republice, získaných z otevřených zdrojů. Práce bude řešit dva cíle. Hlavním cílem je sestavit mapu nejnebezpečnějších oblastí na základě informací získaných během studie. Sekundárním cílem je vytvořit otevřenou databázi dopravních nehod shromážděnou během výzkumu pro její další použití nadšenci a specialisty v oblasti analýzy dat a strojového učení.

**Metodika**

První část práce bude rešerše dostupných zdrojů, použitých metod a nástrojů. V druhé části práce bude dokumentován návrh aplikace pro sběr dat pomocí programovacího jazyka Python. Budou dodržovány standardy obvyklé v softwarovém inženýrství, především modelovací jazyk UML.

**Doporučený rozsah práce**

40 – 80 stran

**Klíčová slova**

dopravní nehody; Python; shluková analýza; mapy; otevřená databáze

---

**Doporučené zdroje informací**

JAIN, A.K, M.N MURTY a P.J FLYNN. Data Clustering: A Review. ACM Computing Surveys [online]. Sept. 1999, 1999(31), 323 [cit. 2020-11-10]. Dostupné z: doi:10.1145/331499.331504

Vision Zero. AHMADI, Negar. Casebook of Traumatic Injury Prevention [online]. 1. Cham: Springer International Publishing, 2020, s. 285-300 [cit. 2020-11-10]. ISBN 978-3-030-27419-1. Dostupné z: https://link.springer.com/book/10.1007/978-3-030-27419-1

ЗАГОРУЙКО, Н.Г. Прикладные методы анализа данных и знаний [online]. 1. Новосибирск: ИМ СО РАН, 1999 [cit. 2020-11-10]. ISBN 5-86134-060-9. Dostupné z: https://www.rfbr.ru/rffi/ru/books/o_18614

---

**Předběžný termín obhajoby**

2020/21 LS – PEF

**Vedoucí práce**

doc. Ing. Vojtěch Merunka, Ph.D.

**Garantující pracoviště**

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 11. 2020

**Ing. Martin Pelikán, Ph.D.**

Vedoucí katedry

Elektronicky schváleno dne 19. 11. 2020

**Ing. Martin Pelikán, Ph.D.**

Děkan

V Praze dne 13. 03. 2021

**Čestné prohlášení**

Prohlašuji, že svou bakalářskou práci "Analýza dopravních nehod na území České republiky" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.03.2021 _____

**Poděkování**

Rád bych touto cestou poděkoval Doc. Ing. Vojtěchu Merunkovi, Ph.D. za odbornou pomoc a cenné rady při zpracování teto diplomové práce a také mé rodině za podporu během celého studia.

# Analýza dopravních nehod na území České republiky

**Abstrakt**

Rostoucí motorizace s sebou přináší problémy spojené s nárůstem dopravních nehod, což má zvláštní dopad na velká města s vysokou hustotou obyvatelstva. V současné době existují osvědčené a ověřené metody ke snížení nehodovosti. Aby však byla jejich implementace úspěšná, je nutné poskytnout výzkumným pracovníkům pohodlný přístup k informacím které se týkají daného tématu. Tato práce je založena na sběru a systematizaci otevřených dat dostupných na internetu a na jejich další analýze za účelem identifikace nejvíce krizových oblastí v České republice. K dosažení tohoto cíle v práci byly použity možnosti programovacího jazyka Python, stejně jako řady speciálních knihoven, rozšiřujících jeho funkčnost.

**Klíčová slova:** dopravní nehody; Python; shluková analýza; mapy; otevřená databáze

# Analysis of traffic accidents in Czech Republic

**Abstract**

Increasing motorization brings with it problems associated with an increase in traffic accidents, which impacts large cities with a high population density. There are currently good and proven methods to reduce accidents. However, in order for their implementation to be successful, it is necessary to provide researchers with convenient access to information related to the topic. This work is based on the collection and systematization of open data available on the Internet and their further analysis to identify the most dangerous areas in the Czech Republic. Python programming language was used, with several special libraries that extend its functionality to achieve this goal.

**Keywords**: traffic accidents; Python; clustering; maps; open database

# Obsah

# Seznam obrázků

# 1 Úvod

V posledních desetiletích dopravní nehodovost na celém světě neustále roste. Je to způsobeno několika různými faktory. Například tím, že dopravní (silniční) infrastruktura zaostává za nároky a potřebami veřejnosti a státu, systém, jehož úkolem je zajištění bezpečnosti silniční dopravy, není dostatečně efektivní a disciplína všech účastníků silniční dopravy je na velmi nízké úrovni.

S tím, jak postupně docházelo k rozvoji automobilového průmyslu, nárůstu množství dopravních prostředků na silnicích a také počtu dopravních nehod, vznikala i potřeba budování systému evidence a analýzy příčin a podmínek vedoucích ke vzniku dopravních nehod. Zároveň s tímto systémem začala být formulována pravidla evidence a klasifikace dopravních nehod.

V roce 2019 bylo v České republice evidováno více než 107 tisíc dopravních nehod, v jejichž důsledku bylo zraněno více než 26 tisíc lidí (Straka, 2020). Zde je nutné zdůraznit, že nehody s fatálními důsledky, se zraněními lidí a ztrátou materiálních hodnot, způsobují významnou sociálně ekonomickou škodu také státu.

Odhad výše ztrát způsobených dopravními nehodami a sdílení této informace s obyvatelstvem mají významný sociální a psychologický účinek. Tyto informace dokáží varovat obyvatele, upozornit je na možné ohrožení jejich života a zdraví a napomáhají tomu, aby si lidé uvědomili důležitost opatření zaměřených a vedoucích ke zlepšení celkové situace na silnicích.

Vůbec poprvé jsem nápad na sběr dat o dopravních nehodách (dále v textu také jako havárie) dostal poté, co jsem si všiml skutečnosti, že neexistuje žádný otevřený a veřejně přístupný zdroj, z něhož by bylo možné data o haváriích ve formě vhodné k jejich další analýze čerpat.

Proč právě dopravní nehody? Za prvé se domnívám, že z důvodu toho, že práce policistů je přísně regulována předpisy, jsou i forma a obsah záznamů o dopravní nehodě, které jsou policisty vyplňovány v místě nehody, také přísně upraveny zákonem. A to znamená, že se nám v případě analýzy velkého množství dat často otevírá možnost stanovení datové domény v konkrétním atributu. Jinak řečeno lze konstatovat, že odpovědi, které uvádí policista v záznamech, jsou často typu „zvolte jednu z možností“ nebo uveďte „ano“ nebo „ne“, což umožňuje data zpracovávat s pomocí speciálních algoritmů, jejichž cílem je hledání skrytých závislostí. Ve výsledku tohoto procesu mohou být získány nové

informace, a tedy i znalosti. Za druhé je nutné konstatovat, že v dnešní moderní době jsme vystaveni vlivu tak obrovského množství informací, že si z důvodu informačního zahlcení přestáváme všímat opravdu důležitých témat. Úmrtnost na silnicích je aktuální problém, který akutně vyžaduje zvláštní pozornost. Zvlášť, když existují již hotová a účinná řešení umožňující s ním účinně bojovat.

# 2 Cíl práce a metodika

## 2.1 Cíle práce

Tato práce se zaměřuje na analýzu údajů o dopravních nehodách v České republice, získaných z otevřených zdrojů. Práce bude řešit dva cíle. Hlavním cílem je sestavit mapu nejnebezpečnějších oblastí na základě informací získaných během studie. Sekundárním cílem je vytvořit otevřenou databázi dopravních nehod shromážděnou během výzkumu pro její další použití nadšenci a specialisty v oblasti analýzy dat a strojového učení.

## 2.2 Metodika

První část práce bude rešerše dostupných zdrojů, použitých metod a nástrojů. V druhé části práce bude dokumentován návrh aplikace pro sběr dat pomocí programovacího jazyka Python. Budou dodržovány standardy obvyklé v softwarovém inženýrství, především modelovací jazyk UML.

# 3 Teoretická východiska

## 3.1 Problém

Není žádným tajemstvím, že automobilová a motocyklová doprava je nejnebezpečnějším typem dopravy, který v současné době existuje. Kromě řidičů dopravních prostředků se oběťmi dopravních nehod stávají často také spolujezdci z dopravních prostředků nebo chodci.

Jak již bylo řečeno výše, stojí nás dopravní nehody ročně kromě vysokých materiálních škod také tisíce lidských životů. A se zrychlující automobilizací a urbanizací nebezpečí roste stále rychlejším tempem.

Vhodným zdrojem pro seznámení se situací ve světě v této oblasti je Světová zdravotnická organizace, z jejíchž údajů je patrné, že problém je opravdu kolosální. Kromě číselných dat umožňuje tento zdroj také získat informace o konkrétních problémech, které trápí jednotlivé státy (World Health Organization, 2018).

V současné době jsou státními orgány jednotlivých zemí v boji s nehodovostí využívány sankce a postihy, které jsou ukládány těm, kteří poruší pravidla silničního provozu. Tato opatření jsou nicméně účinná jen částečně. V následující kapitole bych Vás rád seznámil se zajímavou koncepcí, která vznikla ve Švédsku a je zaměřena na snížení úmrtnosti na silnicích.

## 3.2 Program Vision Zero

Program Vision Zero byl vyvinut v roce 1995 švédským Státním úřadem pro silniční dopravu . Tento projekt vyvolal velký zájem napříč politickým spektrem, zaujal také ministryni dopravy Švédska Ines Uusmann a již dva roky po svém vzniku, v říjnu roku 1997, začal být využíván na státní úrovni.

Základním principem programu je nepřípustnost dopravních nehod s fatálními následky. Tato zásada je také nazývána principem „nulové tolerance" vůči úmrtím na silnicích. Klíčovou je zde teze, že problému musí být věnována pozornost do té doby, než se ho podaří zcela vyřešit. Snížení úmrtnosti je dobrou tendencí a ukazatelem, nicméně sama její existence je znakem toho, že stále existují problémy, které ještě vyřešeny nebyly. Hlavním přístupem programu k tomuto problému je provedení detailní analýzy havárií a rozdělení odpovědnosti za ně nejenom mezi účastníky silničního provozu, ale také mezi lidi, kteří se podílejí na plánování, projektování a uvádění pozemních

komunikací, na nichž k haváriím dochází, do provozu, dále pak mezi výrobce automobilů atd. Vývojáři programu si uvědomují, že účastníci silničního provozu, a to ať už řidiči nebo chodci, jsou jen lidé a lidé dělají chyby. Proto je potřeba se nespoléhat na jejich pozornost a opatrnost, ale vytvářet takové prostředí, které bude lidským chybám předcházet. Přejděme ke konkrétním příkladům. Jedním z hlavních faktorů, které mají vliv na počet a závažnost dopravních nehod, je nedodržování rychlostních limitů. Ukazuje se, že mnohem účinnější při eliminaci tohoto jevu než využití zákazových značek, je například kombinace různých typů silničního povrchu, budování umělých nerovností a zakřivení samotné silnice, což řidičům fyzicky znemožňuje porušovat pravidla silničního provozu (Miles, 2016). Dalším zajímavým nápadem je zmenšení rozměrů a snížení počtu dopravních značek, který popsal Stephen Johnson v článku „Want Less Car Accidents? Remove Traffic Signals and Road Signs“ (Johnson, 2017). To za prvé donutí účastníky silničního provozu být pozornějšími, za druhé ulice zpřehlední a zlepší viditelnost všech účastníků silničního provozu. Odstranění svodidel a bariér podél silnic ve městě se také ukazuje jako efektivní. Neposkytuje totiž řidičům pocit falešného bezpečí a nutí je při jízdě dávat si pozor na možný nečekaný střet s chodcem. Ploty navíc komplikují proces čištění chodníků čisticím zařízením. To v zimním období také zvyšuje riziko zranění. Dále bych rád uvedl i některé další příklady, které tvoří základnu koncepce programu Zero Vision:

1. Neexistuje nic cennějšího než lidský život a zdraví. Pokud existuje způsob vyhnout se smrti nebo vážným zraněním, musí být vždy využit. Nic nemůže ospravedlnit oběti na životech.
2. Všechna rozhodnutí musí být založena na znalostech získaných výzkumem.
3. Použita mohou být pouze ta nejlepší řešení. Nedostatek času a vysoké náklady se nemohou stát důvody pro aplikaci špatných řešení, pokud existují lepší.
4. Před provedením jakýchkoliv změn je nutné provést kalkulaci možných rizik, která jsou s nimi spojena. Je potřeba kalkulovat nejenom s riziky, ale také s charakterem škody.
5. Je potřeba vycházet z toho, že odpovědnost za lidské životy nesou ti, kteří projektovali, konstruovali a uváděli do provozu určitý úsek komunikace.

Program začal být využíván v praxi v roce 1997, kdy na silnicích zahynulo 541 lidí (Lindberg, 2017). Pokud bychom se podívali na statistiku úmrtnosti při dopravních

nehodách za celou dobu používání programu, všimli bychom si postupného snižování počtu obětí dopravních nehod. V roce 2019, přestože automobilizace pokračovala rychlým tempem, zahynulo na silnicích 221 lidí (Transport Analysis, 2020). Program tím dokázal svou účinnost. Velmi brzy ho začaly přejímat i jiné státy, například Nizozemsko, Velká Británie a USA.

## 3.3 Otevřenost a dostupnost informací

Další myšlenkou, kterou jsem přijal za vlastní, je ta, že čím více informací bude veřejně přístupných a čím snazší bude přístup k nim, tím více pozornosti veřejnosti dané informace získají, a tedy budou i častěji využívány. Hodnotnost informací je přímo závislá na tom, jak rychle a jak snadno s nimi bude moci být veřejnost seznámena.

Rozhodl jsem se vnést svůj malý přínos k vyřešení situace na silnicích České republiky tím, že shromáždím data o dopravních nehodách publikovaná na webových stránkách Policie ČR, zpracuji je a převedu do podoby, v níž bude snazší s nimi dále pracovat a analyzovat je, což se může ukázat jako zvlášť užitečné pro badatele, jejichž činnost je spojena s vědními obory jako je statistika, strojové učení a projektování silničních staveb.

# 4 Vlastní práce

## 4.1 Sběr informací

### 4.1.1 Úvod

Ke sběru informací jsem využil programovací jazyk Python v kombinaci s nástrojem pro automatické testování Selenium.

Jak vyplývá z dokumentace (Python Software Foundation, 2021) je Python interpretovaný vysokoúrovňový programovací jazyk s širokou škálu knihovních modulů a poměrně jednoduchou a srozumitelnou syntaxí, což z něj činí mimořádně uživatelsky příjemný nástroj pro rychlou tvorbu menších programů a skriptů, vhodný pro naše účely.

Selenium je sada nástrojů umožňující napodobit standardní chování uživatele konečného zařízení ve webovém prohlížeči. Například vyplňování formulářů, výběr různých možností z nabídky, volbu tlačítek a přechody na odkazy. Jinými slovy lze říci, že tato sada nástrojů dokáže zautomatizovat úkony činěné člověkem (Selenium, 2021).

Doplnění Pythonu o nástroje Selenium mi umožnilo napsat jednoduchý internetový bot (web crawler), který automaticky navštěvoval webové stránky zprostředkovávané webem české policie a shromažďoval z nich informace. Rád bych zde upozornil na to, že daný program navštěvoval výhradně veřejně přístupné webové stránky a činil tak s velkými prodlevami mezi jednotlivými iteracemi tak, aby nedocházelo ke zbytečnému přetěžování serveru.

Rád bych tímto poděkoval Policii ČR za poskytnutí veřejného přístupu k údajům o dopravních nehodách a také za srozumitelné a uživatelsky příjemné rozhraní webové služby (Policie ČR, 2020).

Na samém začátku svého projektu jsem se pokusil navštěvovat webové stránky s pomocí standardních webových knihoven umožňujících navázání spojení prostřednictvím protokolu http. Tato metoda je velmi vhodná jak z hlediska využití možností a výkonu počítače, tak i z hlediska zátěže na síť, protože nevyžaduje součinnost s prohlížečem, umožňuje vynechat zprostředkovatelské webové stránky a kontaktovat přímo finální.

Prvním problémem, s níž jsem se při rozpracování této možnosti setkal, byla skutečnost, že k návštěvě finálních stránek tímto způsobem je nezbytné rozklíčovat, na jakém principu jsou vytvářeny jejich URL adresy. Nehledě na to, že na některé zákonitosti

se mi podařilo přijít, plně pochopit princip se mi nepodařilo, protože veškeré informace, které jsem měl k dispozici, byly získány jen vlastním pozorováním.

Dalším problémem dané metody se ukázala skutečnost, že metoda neumožňovala komunikaci s prvky rozhraní, pouze nahrávala výchozí kód stránky. To bylo pro mou práci kritické, protože při zjišťování informací jsem počítal se stálou komunikací prostřednictvím interaktivního rozhraní. Kromě dalších komplikací při sběru dat mi tato metoda neumožňovala sběr zeměpisné šířky a délky. Pro získání souřadnic bylo potřeba ručně provádět určité manipulace, díky nimž byl uživatel nakonec přesměrován na stránku online map Google Maps (Google, nedatováno), odkud pak teprve mohl zeměpisné souřadnice čerpat. Právě z tohoto důvodu jsem nakonec upřednostnil nástroj Selenium.

Na první pohled se může program zdát příliš komplikovaný, ale po velkém množství pokusů a omylů se právě tato konfigurace ukázala jako nejspolehlivější a nejvíce odolná proti chybám, což bylo pro mě nejdůležitější. Rád bych zde připomenul, že počet webových stránek, které program navštívil, činil více než 70 tisíc, protože jeden web vždy obsahoval informace jen o jedné dopravní nehodě. Pokud k tomu přičteme 70 tisíc přechodů na Google Maps, dojdeme k celkem 140 tisícům přechodům, a to ještě bez započtení zprostředkovatelských webů. Abych předešel přetížení serveru, nastavil jsem intervaly mezi jednotlivými iteracemi na zhruba 8-10 vteřin. Za účelem minimalizace nákladů na projekt jsem použil jednodeskový počítač Raspberry Pi model 3B+. V souvislosti s funkčními omezeními stroje se po několika stech cyklů doba mezi iteracemi začala prodlužovat na minuty, časem dosahovala i několika minut, takže množství pokusů o kontakt se serverem bylo opravdu minimální. Čas od času jsem byl nucen program manuálně restartovat.

Kromě jiných problémů, které jsem v rámci práce na projektu zaznamenal, mi těžkosti způsobovaly také poruchy připojení, kvůli nimž program přestával fungovat. Schválně jsem restartování programu nezautomatizoval, abych měl vždy možnost zjistit příčinu problému s fungováním programu a rozhodnout o jeho opětovném spuštění. Místo toho jsem nastavil funkci automatických notifikací v případě poruchy programu přes chatovací aplikaci. Tuto etapu nebudu popisovat detailně, abych se příliš nevzdálil od hlavního tématu práce. Pokud budete mít zájem, můžete se s kódem programu seznámit podrobněji v příloze dané práce.

Bohužel finální verze programu nemohla fungovat příliš dlouho z důvodu globální aktualizace webu, k níž došlo přibližně 1. prosince 2020. V souvislosti s totálním přepracováním jak jeho struktury, tak i způsobu sdílení informací, se další sběr dat ukázal jako nemožný. Proto jsem k větší části dat, na které se odkazuji na konci této práce a také k následné analýze dospěl na základě údajů, které byly z větší části shromážděny předchozími verzemi programu.

Ve skutečnost to však není zdaleka vše, co se mi podařilo nashromáždit. První verze programu totiž fungovaly mnohem rychleji, protože nevyužívaly Selenium a stihly shromáždit velké množství dat, která se ale ukázala jako nepříliš užitečná, protože neobsahovala zeměpisnou šířku a délku, a to z důvodů, které jsem již popsal výše.

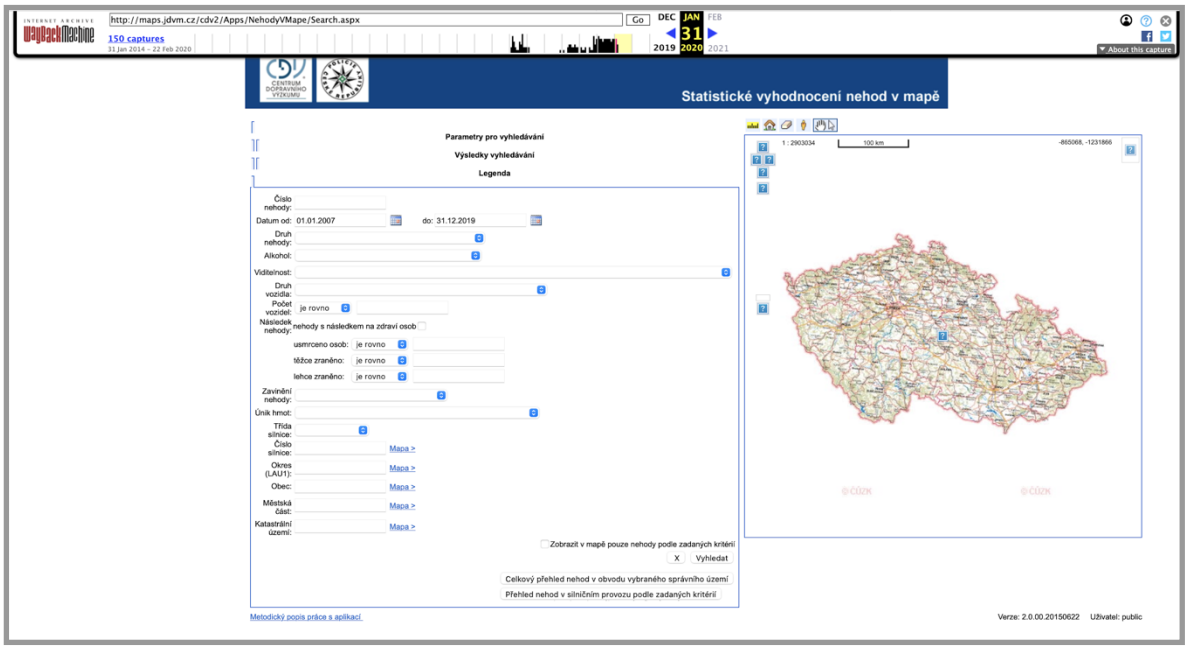
Domnívám se, že počínaje tímto okamžikem bude pro zjednodušení a zpřístupnění informací obsažených v dané práci rozumné text doprovodit obrázky.

Pro začátek bych Vás rád stručně seznámil se strukturou webu. Bohužel z důvodu rozhodnutí web dále neudržovat jsem musel jeho strukturu obnovit podle uložených snímků obrazovky, které byly pořízeny při průzkumu webu, vlastního popisu a s pomocí služby Web Archive (Internet Archive, 2021) umožňující získat přístup k uloženým „snímkům“ webů, které byly pořízeny v minulosti. V mém případě Web archive umožňuje spatřit jen hlavní stránku, ale bohužel již neumožňuje přechod na další stránky s ní spojené. Je to dáno tím, že stránky poskytnuté webovou službou policie nebyly statické, nýbrž byly generovány v závislosti na detailech žádosti vždy individuálně pro každého uživatele. Nicméně i tak se vynasnažím co nejsrozumitelněji popsat postupy, s jejichž pomocí jsem postupně dospíval k jednotlivým řešením a popíšu, s čím byla rozhodnutí spojena.

### 4.1.2 Struktura webu

Na snímku obrazovky níže si můžete prohlédnout hlavní stránku portálu. Hlavní stránka nabízela možnost výběru kritérií, podle nichž budou filtrovány výsledky vyhledávání.

Jak je patrné ze snímku, mohl si uživatel vybrat jak konkrétní dopravní nehodu, jež ho zajímala tím, že uvedl její *unikátní číslo*, tak i pracovat s celou skupinou havárií tak, že dané pole nevyplnil.




*Obrázek 1.* Snímek *hlavní stránky portálu*. Zdroj: *(Policie ČR, 2020) (uloženo ve webovém archivu).*

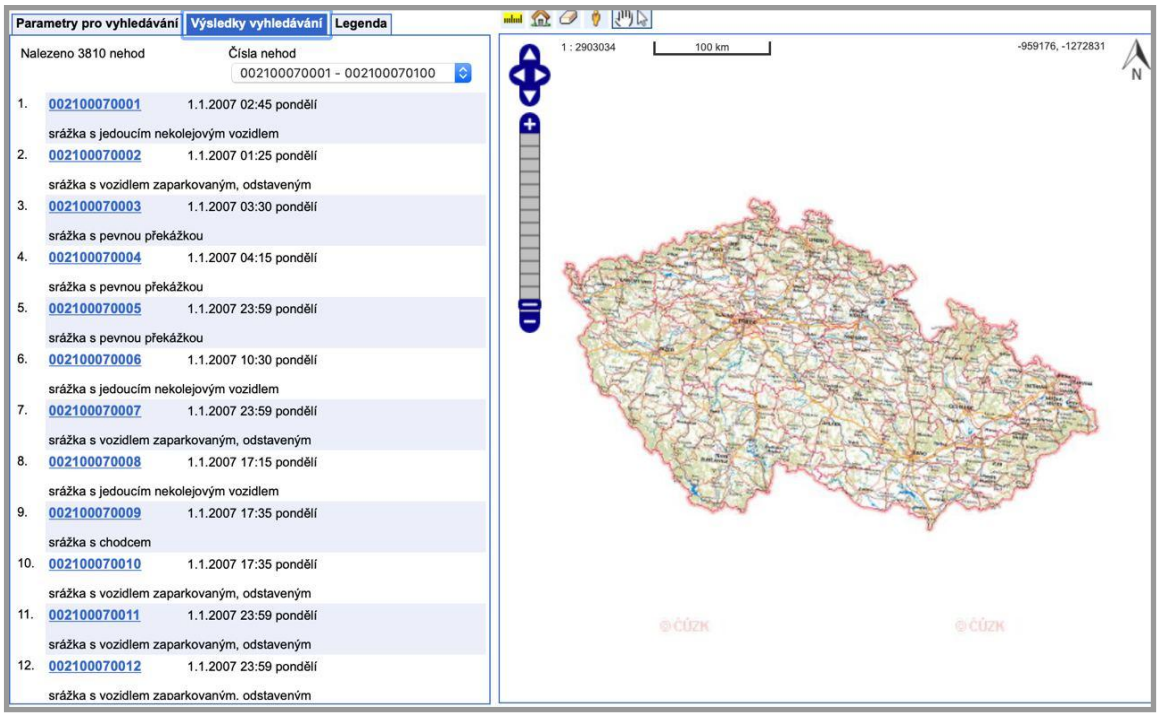
Níže se nachází seznam kritérií, které bylo možné využít jak samostatně, tak i v kombinaci. Pro získání výsledků odpovídajících zvoleným kritériím se od uživatele očekávalo, že stiskne tlačítko „vyhledat“, které se na obrázku nachází v pravém dolním rohu levého okna.

V rámci dané etapy je nutné vysvětlit jakým způsobem programátor dává příkazy Seleniu k provádění určitých úkonů na webové stránce.

```
<input type="submit" name="TcoMain$TpaObject$BtnSearch" value="Vyhledat"
onclick="javascript:WebForm_DoPostBackWithOptions(new
WebForm_PostBackOptions(&quot;TcoMain$TpaObject$BtnSearch&quot;,
&quot;&quot;, true, &quot;&quot;, &quot;&quot;, false, false))"
id="TcoMain_TpaObject_BtnSearch" title=" [search_btn101]"
class="btnSearch"/>
```

Pokud se například podíváme na HTML kód bezprostředně odpovídající za funkci tlačítka „vyhledat“, uvidíme, že se vyznačuje určitým počtem atributů, například `"value"`, `"id"` a `"class"`. Zde můžeme využít Selenium a s jeho pomocí nalézt a provést virtuální „stisk“ prvku, jehož id se rovná `"TcoMain_TpaObject_BtnSearch"`. Jinak řečeno stisknout tlačítko „vyhledat“.

Je důležité si uvědomit, že **hodnoty** atributů mohou být jak unikátní, tak i opakující se. Často to lze pochopit již z názvu atributu. Například atribut „class“ se vztahuje k celé řadě objektů, zatímco atribut „id“ je pro stránky dané webové služby unikátní. Proto pokud potřebujeme spolupracovat s řadou objektů na stránce, musíme využít atribut class, zatímco při vyhledávání konkrétního objektu musíme při vyhledávání použít atribut id.
K tomu, jakým způsobem byl realizován mechanismus projektu, se vrátíme později. Nyní bych rád popsal standardní relaci součinnosti s webovou službou za účelem vytvoření šablony chování uživatele daného zdroje. Vybrali jsme tedy kritéria, která nás zajímala, stiskli tlačítko vyhledávání a následně byli přesměrováni na stránku viz níže.




*Obrázek 2. Příklad výsledků vyhledávání. Zdroj: (Policie ČR, 2020).*

Nyní vidíme seznam odkazů na stránky o konkrétních dopravních nehodách. Číselný kód, který je odkazem, je zároveň také **unikátním** identifikátorem konkrétní dopravní nehody. Je to důležitý detail, protože tento identifikátor se účastní vytváření URL adresy webové stránky, které odpovídá. V případě, že existuje více než 100 výsledků splňujících kritéria našeho vyhledávání, objeví se v pravém horním rohu levého okna rozbalovací nabídka. Nabídka umožňuje přejít na další webové stránky s výsledky. Tyto stránky jsou dále členěny do skupin vždy maximálně po 100 exemplářů. Výběrem unikátního identifikátoru

je uživatel přesměrován na konečnou stránku obsahující podrobné informace o konkrétní dopravní nehodě.

**Charakteristiky řidiče vozidla a příčiny nehody**

| | | | |
|---|---|---|---|
| Zavinění nehody | řidičem motorového vozidla | Alkohol u viníka nehody | ne |
| Kategorie řidiče | s řidičským oprávněním skupiny b | Stav řidiče | dobrý -žádné nepříznivé okolnosti nebyly zjištěny |
| Vnější ovlivnění řidiče | řidič nebyl ovlivněn | | |

**Charakteristiky následků osob - stav do 24 hod**

| | | | |
|---|---|---|---|
| Usmrceno osob (počet) | 0 | Těžce zraněno osob (počet) | 0 |
| Lehce zraněno osob (počet) | 0 | | |

**Charakteristiky vozidla, viníka nehody a následků nehody na vozidle**

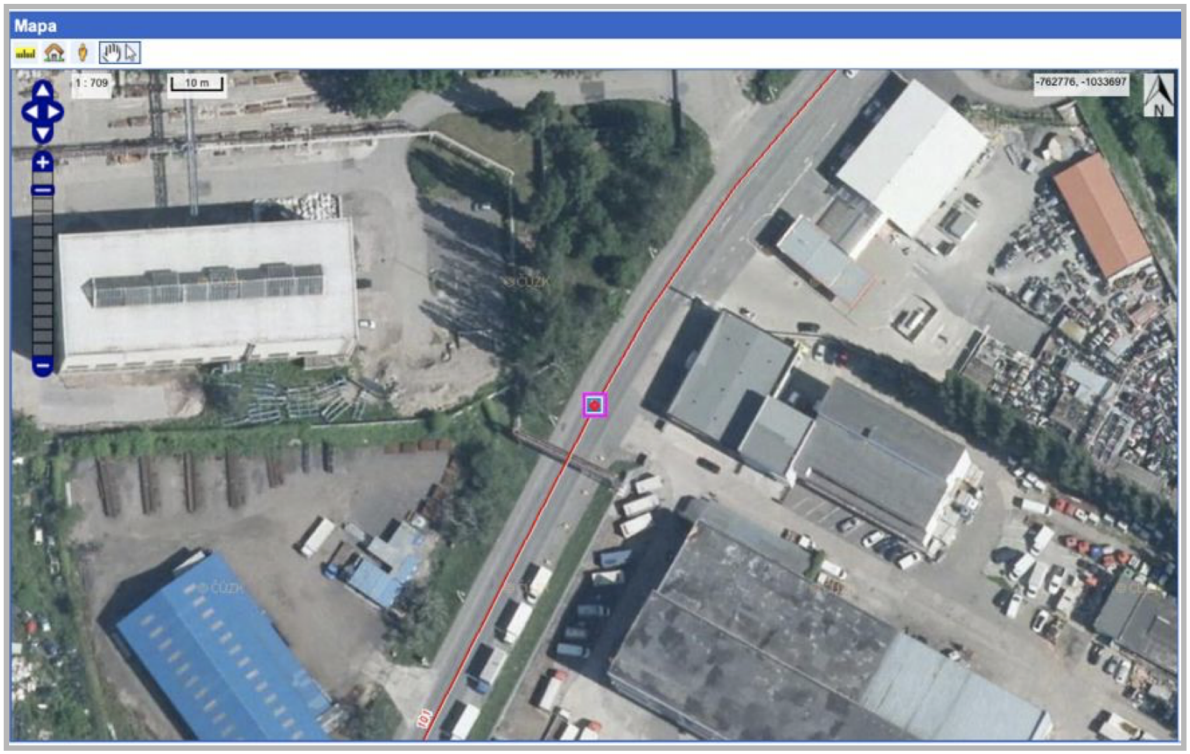
| | | | |
|---|---|---|---|
| Počet zúčastněných vozidel | 2 | Druh vozidla | osobní automobil bez přívěsu |
| Výrobní značka motorového vozidla | CITROEN | Rok výroby vozidla | 03 |
| Vozidlo po nehodě | nedošlo k požáru | Vlastník vozidla | soukromá organizace (podnikatel, s.r.o., v.o.s., a.s.,atd.) |
| Celková hmotná škoda (100 Kč) | 100 | Škoda na vozidle (100 Kč) | 0 |
| Únik provozních, přepravovaných hmot | žádné z uvedených | Způsob vyproštění osob z vozidla | nebylo třeba užít násilí |

**Charakteristiky druhu nehody a podmínek nehody**

| | | | |
|---|---|---|---|
| Charakter nehody | nehoda pouze s hmotnou škodou | Druh nehody | srážka s jedoucím nekolejovým vozidlem |
| Druh srážky jedoucích vozidel | zezadu | Druh pevné překážky | nepřichází v úvahu, nejde o srážku s pev.překážkou |
| Hlavní příčiny nehody | nedodržení bezpečné vzdálenosti za vozidlem | Druh povrchu vozovky | dlažba |
| Stav povrchu vozovky v době nehody | povrch mokrý | Stav komunikace | dobrý, bez závad |
| Povětrnostní podmínky v době nehody | neztížené | Viditelnost | v noci - s veřejným osvětlením,viditelnost nezhoršená vlivem povětrnostních podmínek |
| Rozhledové poměry | dobré | | |
| Situování nehody na komunikaci | na jízdním pruhu | Dělení komunikace | žádná z uvedených |
| Místní úprava přednosti v jízdě | žádná místní úprava | Řízení provozu v době nehody | žádný způsob řízení provozu |
| Směrové poměry | zatáčka | Specifické objekty v místě nehody | žádné nebo žádné z uvedených |
| Druh křižující komunikace | neurčeno | Místo dopravní nehody | mimo křižovatku |
| Směr jízdy nebo postavení vozidla | jedoucí - ve směru staničení na komunikaci | Smyk | ne |

*Obrázek 3. Popis dopravní nehody.* Zdroj: *(Policie ČR, 2020).*

Na stránce lze také nalézt vloženou mapu, na níž bude označeno místo, v němž ke konkrétní dopravní nehodě došlo. Nicméně, pokud bychom se pozorně podívali na souřadnice, které se nacházejí v pravém horním rohu, pochopili bychom, že nemají nic společného s obvyklým formátem zeměpisné délky a šířky.




*Obrázek 4. Vložená mapa. Zdroj: (Policie ČR, 2020).*

Skutečné zeměpisné souřadnice lze získat, pokud vybereme piktogram člověčka v levém horním rohu rozhraní vložené mapy a následně vybereme červený bod na mapě. Tehdy bude uživatel přesměrován na identické místo, ale již na Google Maps, kde nalezne skutečné souřadnice. A právě v tomto místě jsem zaznamenal problémy. Nepodařilo se mi totiž v kódu mapy vložené na web nalézt objekt, na který bych mohl pro kliknutí zaměřit Selenium. Ale všiml jsem si toho, že mapa je vždy vycentrována podle místa, v němž došlo k dopravní nehodě. Pro podobné situace má knihovna speciální metodu, která umožňuje umístit prvek a kliknout na určitý bod. V našem případě jde o kliknutí do středu mapy.

Zbývalo mi tedy pouze nahrát kódy obou webových stránek a získat z nich informace, které se bezprostředně týkaly dopravní nehody. Za tímto účelem jsem využil možností knihovny Beautiful Soup (Richardson, 2020). Jak vyplývá z jejího popisu, je Beautiful Soup knihovnou Pythonu, která slouží k extrakci dat z HTML a XML souborů, podporující velké množství parserů, schopnou programátorovi ušetřit hodiny a dny práce. Součinnost s Beautiful Soup probíhá podobně jako práce se Seleniem. Od uživatele je vyžadováno pouze uvést znak pro vyhledávání, a to ať už unikátní nebo ne. Následně ho Beautiful Soup vyhledá a extrahuje z něj příslušné hodnoty.

Takto jsem tedy popsal mechanismus, jak shromáždit data o jedné dopravní nehodě. Dále bylo potřeba zautomatizovat proces návštěvy všech případů dopravních nehod.

Navrhuji se na analýzu tohoto mechanismu podívat v obráceném pořadí.
Již máme k dispozici informace o jedné dopravní nehodě. Nyní se musíme vrátit na předcházející stránku a vybrat další dopravní nehodu a zopakovat výše popsaný algoritmus. Tento proces je nezbytné opakovat až do té doby, dokud nebudou zpracovány všechny případy, které jsou na stránce zobrazeny (jak již víme, jedná se až o 100 případů). Následně musíme přejít na další stránku v rozbalovací nabídce (pokud existuje) a opakovat celý již nám známý postup.

Další problém spočívá v tom, že počet prvků rozbalovací nabídky je také omezen sto jednotkami. Což znamená, že najednou můžeme pracovat maximálně s 10 000 případy nehod. První havárie, které jsou evidovány systémem, jsou datovány do roku 2007. Od tohoto okamžiku bylo v systému zaregistrováno více než 1 400 000 nehod. Z toho vyplývá, že abychom mohli shromáždit data o více než 10 tisících z nich, budeme muset také spolupracovat s hlavní stránkou, stránkou vyhledávání.
Po velkém množství pokusů jsem zvolil variantu postupného procházení dnů vždy po jednom. To mi umožnilo obejít omezení 10 000 případů. Nicméně zde opět vznikly potíže z důvodu specifické konfigurace pole data, využívaného webem. Musel jsem nastavení pole prostudovat a metodou pokusů a omylů zjistit, v jakém formátu je nutné s pomocí Selenia předávat symboly, aby bylo pole data na webu korektně vyplňováno. Nevidím nicméně smysl se zde rozepisovat o technických detailech. Pokud budete mít zájem, můžete si samostatně prostudovat kód.

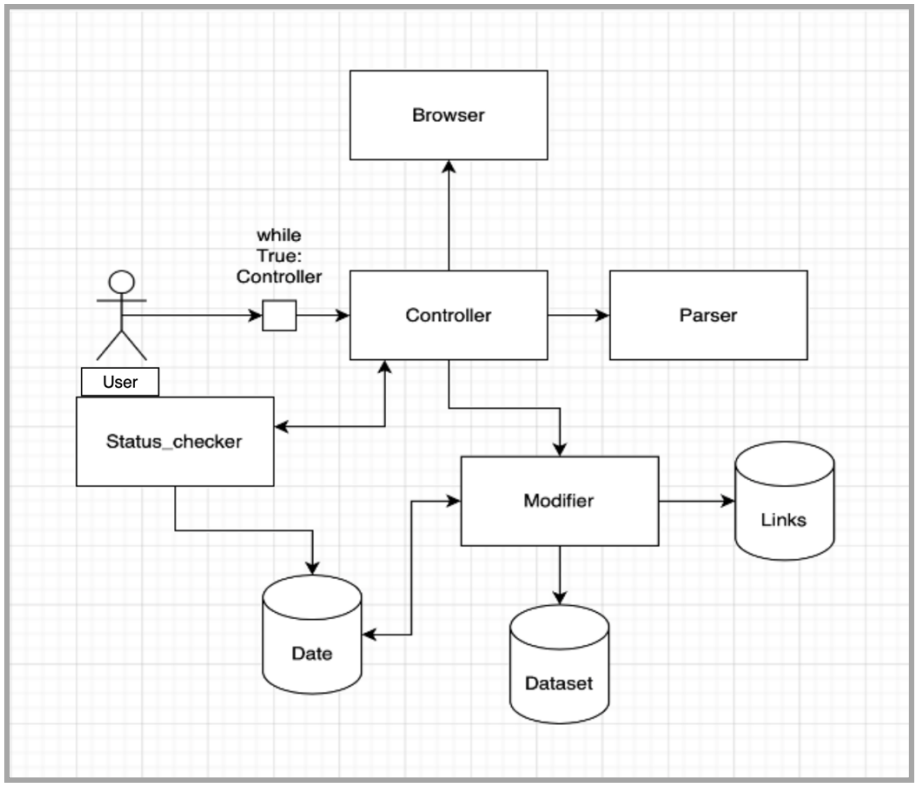
Zbývá tedy pouze vymyslet algoritmus, který by mohl obnovit provoz programu v případě jeho nouzového přerušení. Na něj se za účelem lepší srozumitelnosti podíváme detailněji v kapitole, která se zabývá modelováním programu.

Shrnutí závěrů: program musí dotazovat vyhledávací stránku webu, následně kontrolovat, z jakého dne naposledy zpracovala data a na základě toho vyplňovat formuláře vyhledávacích kritérií. Dále musí program umět roztřídit odkazy odpovídající kritériím, počínaje tím, který následuje za posledním úspěšně staženým odkazem. Přechod na odkaz symbolizující konkrétní dopravní nehodu musí být doprovázen přesměrováním na mapy Google Maps a zkopírováním výchozích kódů obou stránek (stránka s informacemi o dopravní nehodě a stránka map), z nichž musí být následně extrahovány

informace, které uživatele zajímají. Jakmile odkazy dojdou, je nutné zkontrolovat existenci rozbalovací nabídky a v případě, že existuje, zopakovat tento postup pro každou stránku. Nakonec můžeme celý popsaný proces zopakovat pro následující den.

## 4.2 Struktura programu, model

Níže, ve formě UML diagramu, je znázorněno schéma, které považuji za nejsrozumitelnější a nejlogičtější. Na základě tohoto schématu jsem později napsal program. Všechny diagramy byly vytvořeny pomocí nástroje diagrams.net (diagrams.net, 2020)



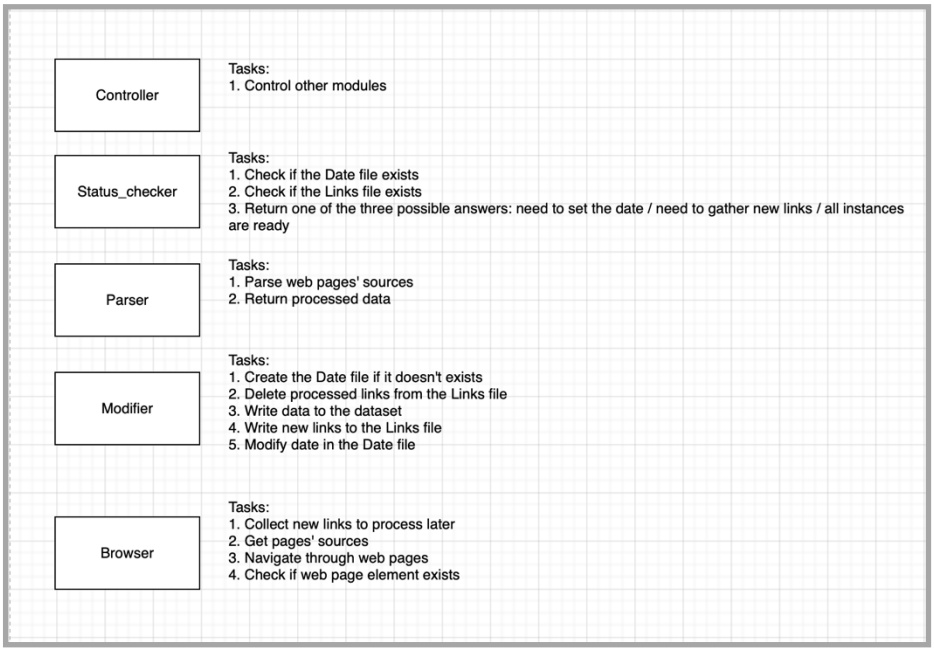

*Obrázek 5. Schéma programu. Zdroj: autor.*

Jak je patrné z obrázku, je program rozdělen na několik modulů, které spolu vzájemně komunikují. Toto řešení bylo zvoleno hlavně kvůli zjednodušení ladění a kontroly provozu programu. Každý modul má svůj vlastní a přesně definovaný účel.

Netriviální výzvou se ukázalo vymyslet systém udržení stavu programu. Za tímto účelem jsem musel výše popsaný přístup poněkud upravit.

Především jsem založil zvláštní soubor pro uložení posledního dne (datum), které program zpracoval a také další soubor, který mi umožnil vzdát se varianty opakovaného

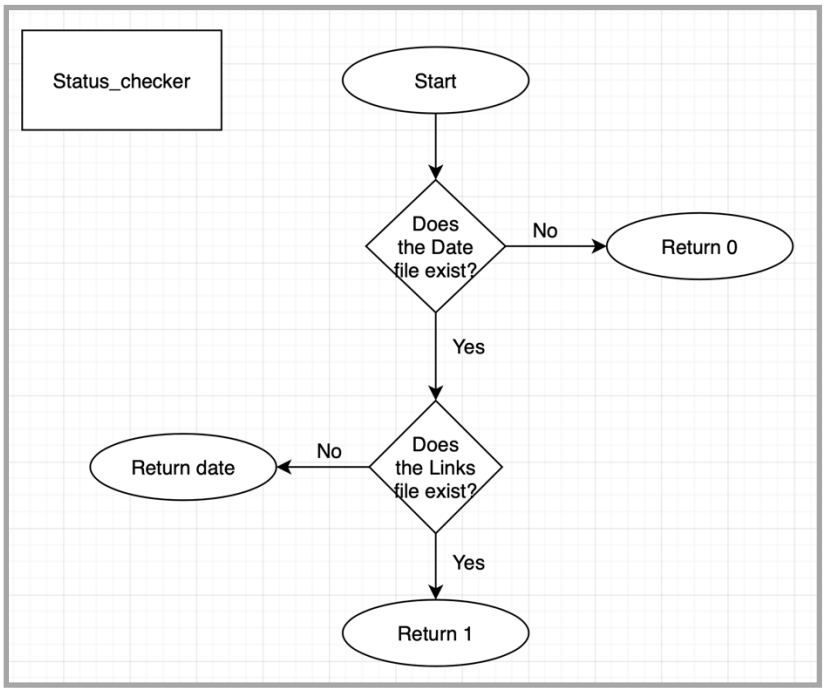
kontaktování webové stránky s výsledky vyhledávání, a to tak, že jsem všechny odkazy uložil do zvláštního souboru a dále pak pracoval bezprostředně jen s ním. Tento způsob mi umožnil nejenom snížit počet požadavků (dotazů), ale i ukládat poslední stav pro případ neočekávané poruchy programu. Dále to bude demonstrováno názorně. Nyní bych chtěl popsat všechny moduly programu podrobněji.




*Obrázek 6. Popis modulů. Zdroj: autor.*

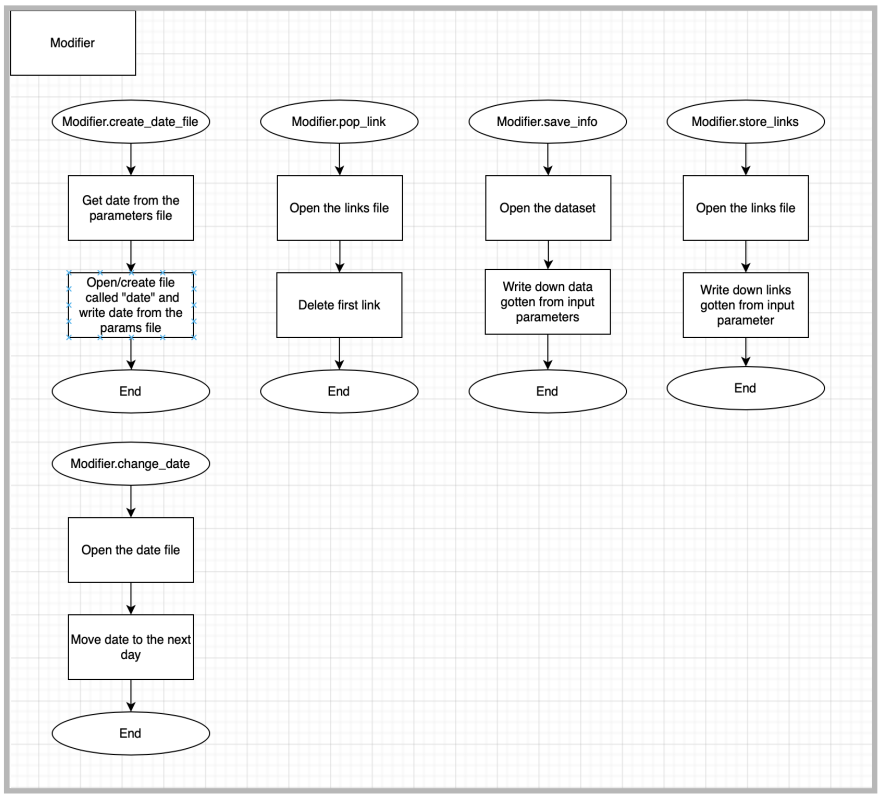
Jediné, co program vyžaduje po uživateli je určení příčiny poruchy programu v případě chybového hlášení a manuální restartování programu. Následují programové moduly.

Začněme nejjednodušším modulem status_checker.py. Jeho úkolem je kontrola existence souborů date a links a kontrola jejich obsahu, tj. ujištění se, že nejsou prázdné, o čemž modul vrací odpověď. Odpověď může mít jeden ze tří významů, buď „date neexistuje“ nebo „soubor links bez odkazů“ nebo „připraveno na provoz“. Na základě výsledku zjištěného modulem status_checker.py přijímá hlavní modul rozhodnutí o tom jaké úkony budou dále provedeny.




*Obrázek 7. Schéma modulu status_checker.py. Zdroj: autor.*

Následuje modul modifier.py. Odpovídá za práci se soubory, mění data v nich uložená.



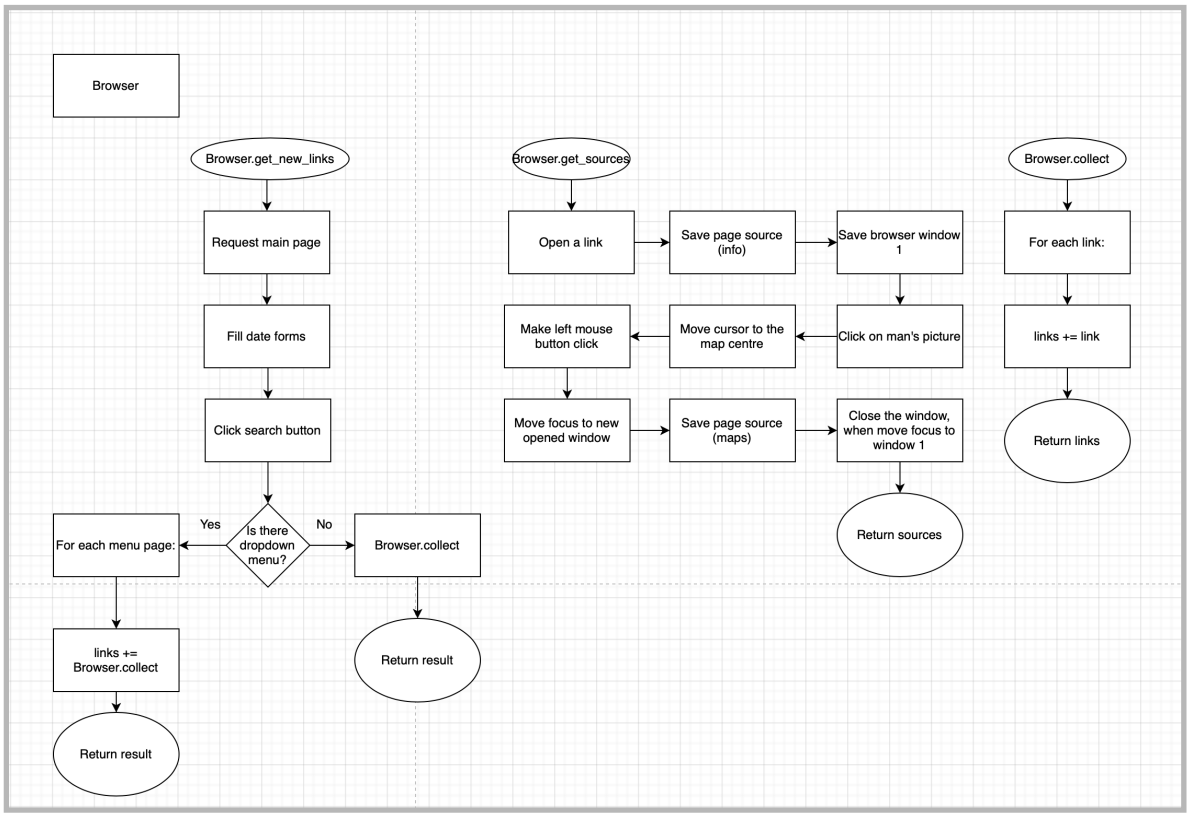

*Obrázek 8. Schéma modulu modifier.py. Zdroj: autor.*

Jak je patrné z obrázku výše, skládá se modul z pěti funkcí, které vykonávají maximálně jednoduché úkoly.
Konkrétně tyto:

- V případě, že soubor, který by měl obsahovat údaje o posledním zpracovaném dni, neexistuje, bude soubor vytvořen a doplněn o výchozí hodnotu, tedy den, kterým jsou datovány první záznamy o dopravních nehodách.
- V případě, že byla zaznamenána žádost o modifikaci souboru links, modifier tento soubor otevře a odstraní z něj první odkaz v pořadí.
- Modul parser.py po extrahování dat z výchozích kódů stránek předá tato data modifieru s požadavkem na jejich nahrání do datové sady.
- Modul browser v případě, že se ukáže, že soubor links je prázdný, obdrží nové odkazy ke zpracování a předá je modulu modifier s požadavkem na jejich nahrání do souboru links.
- Modul controller.py donutí modifier převést datum v souboru date na následující den, pokud byly všechny případy dopravních nehod za aktuální den již zpracovány.

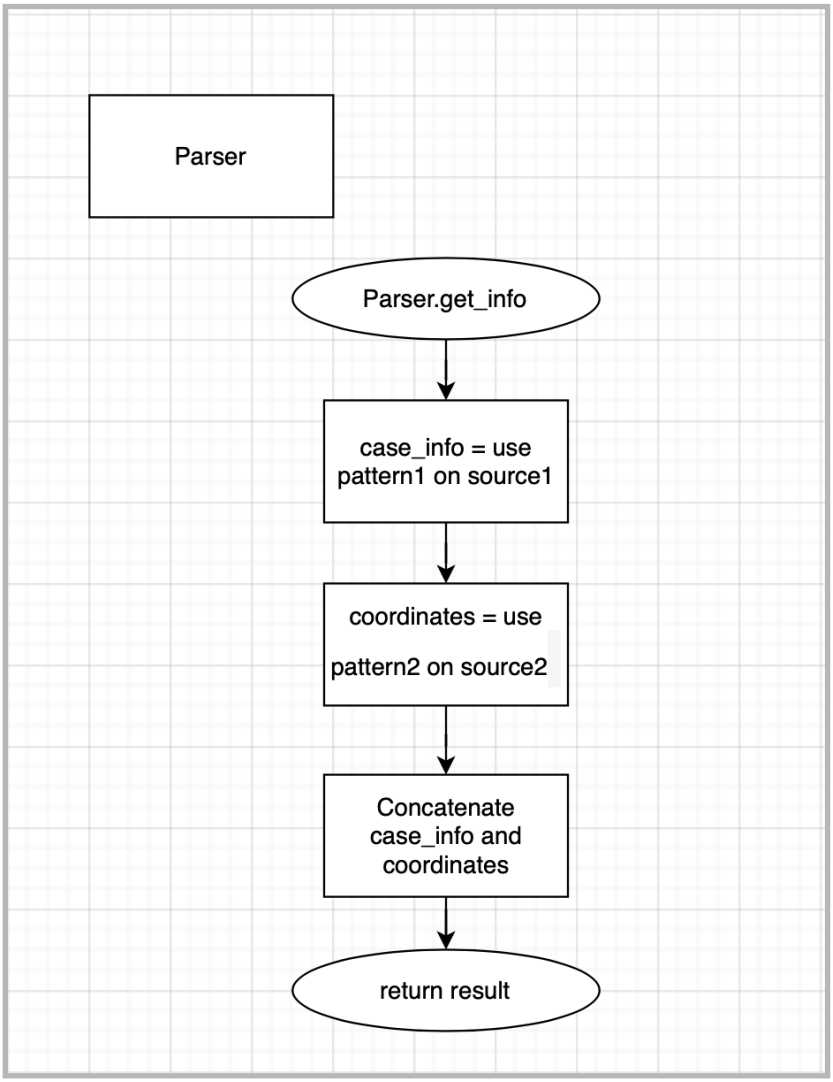
Modul my_browser.py je odpovědný za komunikaci s driverem prohlížeče. Také umí vykonávat několik dalších úkolů. Tentokrát poněkud složitějších:

- Za prvé umí shromažďovat odkazy k dalšímu zpracování, a to s pomocí metody get_new_links.
- Za druhé s pomocí metody get_sources umí (pokud dostane jako argument odkaz na stránku) zažádat o její výchozí kód a následně získat stránku z Google Maps, která odpovídá dané konkrétní dopravní nehodě a také extrahovat její výchozí kód.
- Metoda collect je pomocná metoda metody get_new_links.




*Obrázek 9. Schéma modulu browser.py. Zdroj: autor.*

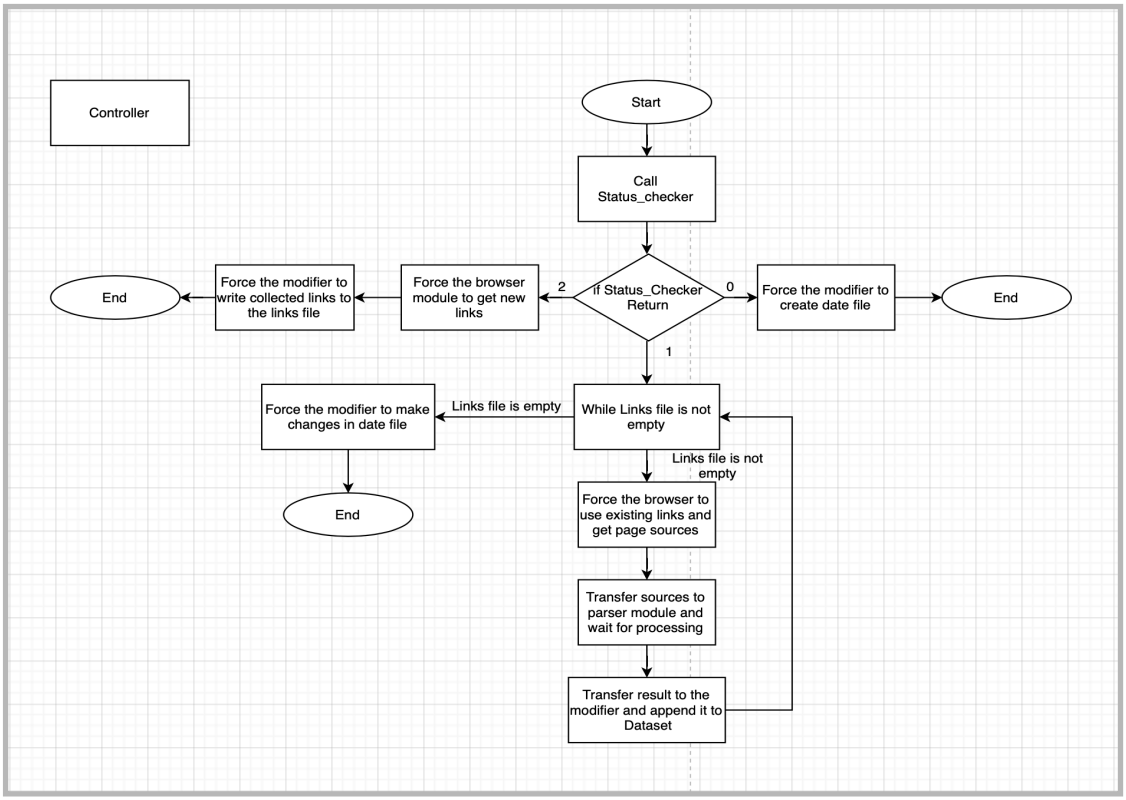
Dále následuje modul parser. Ten dostává jako argument výchozí kód dvou stránek, z nichž zpracuje informace za pomoci předem připravených šablon (regulárních výrazů) a následně informace získané z obou stránek sloučí, vrátí je a předá k nahrání do datové sady.




*Obrázek 10. Schéma modulu parser.py. Zdroj: autor.*

A nakonec modul, který je odpovědný za řízení celého procesu - controller.py.

Díky rozdělení programu na části se podařilo vyvinout maximálně jednoduchý program a zároveň vyloučit situaci, při níž by po restartování, k němž došlo z důvodu neočekávaného ukončení programu, program vynechal nebo zkopírovat část dat.




*Obrázek 11. Schéma modulu controller.py. Zdroj: autor.*

Pojďme nasimulovat standardní relaci provozu programu.

Po spuštění začne program cyklicky dotazovat controller.py. Controller nejdříve dotáže modul kontroly statusu, který odpovídá za kontrolu stavu, v němž byla ukončena předcházející relace programu (pokud existovala). Předpokládejme, že se jednalo o první spuštění. V tomto případě soubor obsahující datum, od něhož musíme zahájit sběr dat, ještě nebyl vytvořen, proto status_checker odpoví hodnotou 0. Jakmile controller obdrží tuto odpověď, vybere metodu modulu modifier, aby mohl být vygenerován soubor date a následně se ukončí. Vnější cyklus opět spustí controller.py. Tentokrát již status_checker po dokončení kontroly vrátí hodnotu 2. V reakci na to controller zvolí modul odpovídající za práci s prohlížečem a vydá příkaz k naplnění souboru links odkazy, které budou později zpracovány. Prohlížeč poté, co se dostane na hlavní stránku, vyplní povinná pole a přejde k výsledkům vyhledávání. Následně zkontroluje, zda existuje rozbalovací nabídka a zpracuje všechny odkazy, které jsou ve výsledcích vyhledávání. Dále z nich vytvoří jednotnou strukturu a vrátí je controlleru. Controller předá data, která získal, modifieru za účelem jejich nahrání do souboru links. Poté, co budou data nahrána do souboru, práce končí. Vnější cyklus opět vyvolá controller. Tentokrát se status_checker ujistí, že je vše připraveno k provozu a odešle controlleru odpověď 1. Na což controller zareaguje

následovně: začne cyklicky třídit odkazy ze souboru links a postupně je předávat metodě modulu browser, který je odpovědný za získání výchozích souborů. Jakmile obdrží odkaz, požádá prohlížeč o odpovídající stránku a zapamatuje si její výchozí kód, následně pak přejde na s ní spojenou stránku Maps a také si zapamatuje její výchozí kód. Shromážděná data opět vrátí controlleru, který je přepošle parseru, aby z nich mohly být extrahovány užitečné informace. Poslední etapou bude předání získaných informací modulu modifier za účelem jejich nahrání do datové sady a odstranění právě zpracovaného odkazu ze souboru links.

Příkladem výsledku práce jedné iterace programu je následující řádek:

```
"002100145671"   "Praha (Hlavní město Praha)" "24.4.2014 17:10 čtvrtek"
     "komunikace místní"    "0"   "řidičem motorového vozidla" "s
řidičským oprávněním skupiny b"    "řidič nebyl ovlivněn" "ne"  "dobrý -
žádné nepříznivé okolnosti nebyly zjištěny"    "0"   "0"   "0"   "2"
     "CITROEN"   "nedošlo k požáru"     "350" "žádné z uvedených"
     "osobní automobil bez přívěsu"     "07"  "soukromé, nevyužívané k
výdělečné činnosti"    "50"  "nebylo třeba užít násilí"   "nehoda pouze
s hmotnou škodou" "zezadu"    "nesprávné otáčení nebo couvání"   "povrch
suchý, neznečistěný"   "neztížené" "dobré"    "na jízdním pruhu"
     "žádná místní úprava"   "přímý úsek"     "neurčeno"  "vozidlo
jedoucí - na komunikaci bez staničení"   "srážka s jedoucím nekolejovým
vozidlem"   "nepřichází v úvahu, nejde o srážku s pev.překážkou" "živice"
     "dobrý, bez závad"     "ve dne, viditelnost nezhoršená vlivem
povětrnostních podmínek"     "dvoupruhová"    "žádný způsob řízení
provozu"    "žádné nebo žádné z uvedených"          "mimo křižovatku"
     "ne"  "14.415680559497785" "50.126196441523604".
```

Jsou v něm postupně uvedeny všechny údaje, které bylo možné zjistit z webových stránek, o nichž jsem již dříve hovořil. Dále na konci řádku vidíme souřadnice.
Každá následující iterace doplní do souboru podobný řádek a struktura dat bude stále rozsáhlejší.

Díky přidaným pomocným souborům uchovávajícím status aktuálního stavu při každé iteraci a také díky tomu, že záznam dat byl prováděn naráz, a ne po jednotlivých etapách, se povedlo dosáhnout odolnosti vůči chybám spojeným s poruchami internetového připojení. V případě vnějších incidentů majících za následek přerušení

fungování programu, dokáže program automaticky nalézt poslední etapu, v níž byl pracovní proces přerušen a bude v další práci pokračovat přesně z tohoto místa.

Shromážděná data jsou k dispizici na https://docs.google.com/spreadsheets/d/1-kDzfA60xtHm6Aq9aNh8zcwMbEdb7RLMy36_JjvtYIs/edit?usp=sharing.

.

## 4.3 Analýza dat

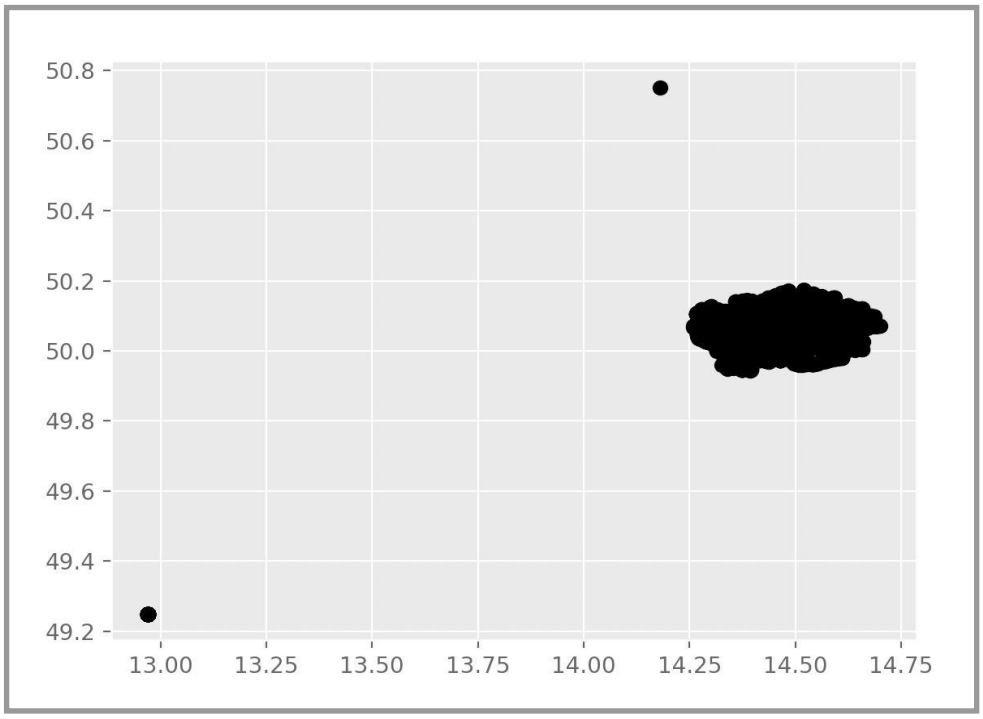
Nyní se podíváme na proces analyzování dat.

Prvním krokem, který jsem učinil po etapě sběru dat, bylo vyhledání duplikátů a neúplných záznamů, které by v budoucnu mohly negativně ovlivnit kvalitu získaných výsledků. Podobné úlohy lze řešit poměrně rychle díky již hotovým nástrojům knihovny Pandas (Pandas, nedatováno) jež je jedním z nejčastěji používaných nástrojů pro práci s podobnými strukturami dat.

Mým dalším úkolem byla kontrola dat, zda neobsahují anomálie. Pro dosažení mého cíle jsou pro mě podstatné pouze zeměpisné souřadnice dopravní nehody, zabýval jsem se tedy pouze jimi.

Abych dokázal nalézt případné anomálie, promítnul jsem zeměpisné souřadnice, které se mi podařilo nashromáždit, na souřadnicové osy, čímž jsem získal mapu dopravních nehod. To mi umožnilo případné anomálie okamžitě zpozorovat.

Jak je patrné z obrázku níže, došlo k většině dopravních nehod, o nichž se mi podařilo získat data, v Praze a jejím okolí. Nehody jsou hustě zkoncentrovány. Je zajímavé, že krajní dolní levý bod leží již za hranicemi ČR. Zatímco horní krajní bod je ještě na území České republiky. Nicméně oba dva krajní body se nacházejí poměrně daleko od většiny ostatních bodů. Protože tyto anomálie tvoří jen malou část všech dat, ale mohou se významně podepsat na výsledku, rozhodl jsem se je vyřadit ze zkoumané skupiny.




*Obrázek 12. Promítnutí bodů na souřadnicové osy. Zdroj: autor.*

Výsledek po odstranění nežádoucích dat vypadal následovně. Přestože množství shromážděných informací tvořilo pouze malou část všech informací uložených ve webové službě, umožnila jejich vizualizace vykreslení obrysů městských částí a ulic.


*Obrázek 13. Centrum města. Zdroj: author.*

Finálním krokem se stalo stanovení ohnisek dopravních nehod.

Základem metod analýzy dat je hypotéza kompaktnosti, která říká, že realizace stále stejného obrazu se obvykle odrážejí v příznakovém prostoru do geometricky blízkých bodů, kde vytvářejí kompaktní shluky (Zagorujko, 1999) Stejně je tomu u hypotézy o lambda kompaktnosti, která na rozdíl od hypotézy kompaktnosti operuje ne s absolutními, ale s relativními vzdálenostmi mezi body (Zagorujko, 1999). Je nezbytné zdůraznit, že lidské oko se dokáže mimořádně dobře vypořádat s úkoly kladenými na něj taxonomií, pokud jsou objekty zastoupeny ve formě bodů v rovině. Lidské oko totiž zaznamenává změny ve stejnorodosti vzdáleností a tím dělí obraz na jednotlivé útvary. Jednoduchým příkladem je obrázek 12, při pohledu na nějž, dokážeme bez problémů rozpoznat 3 skupiny bodů.

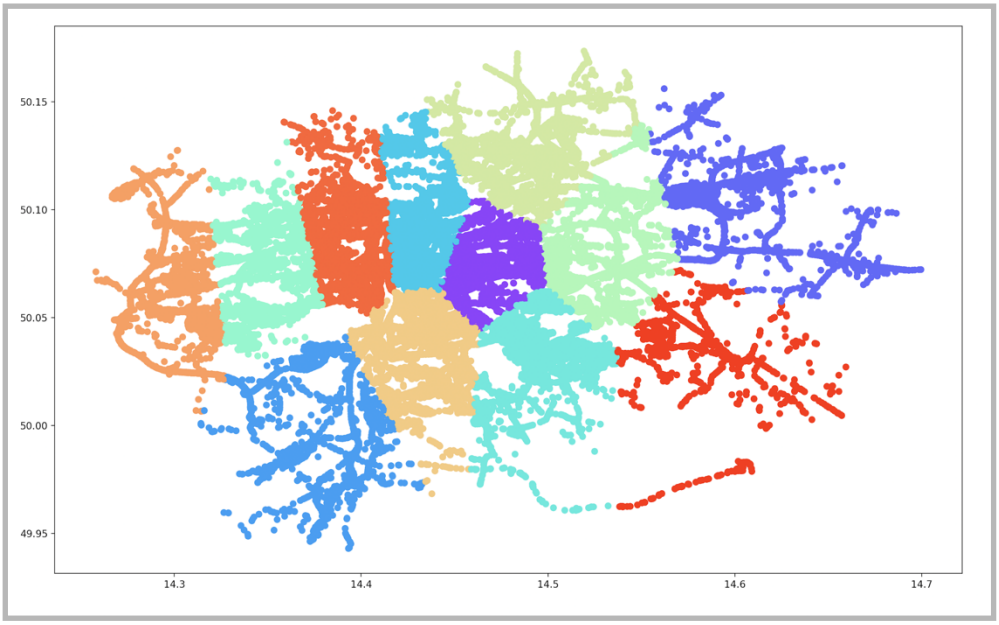
Pokud bychom použili hypotézu kompaktnosti, nepodařilo by se nám získat podobné výsledky, zatímco hypotéza lambda kompaktnosti založená na vztazích mezi vzdálenostmi, dokáže získané výsledky rozklíčovat (Zagorujko, 1999).
Prvním pokusem o extrakci nových znalostí byla aplikace algoritmu k-means. Bohužel pro dosažení cíle, který jsem si vytyčil, se tento algoritmus ukázal jako nepříliš účinný, protože se zaměřuje na *sektory* s nejvyšší hustotou, ne na body. Nicméně i toto řešení může být užitečné, například v případě, že je potřeba nalézt nejvýhodnější umístění pro policejní útvary a rozdělit mezi ně „zóny kompetencí“, za které budou jednotlivé útvary odpovídat. Proto výsledky použití tohoto algoritmu uvedu níže také.

### 4.3.1 Klasterizace, k-means

K-means je algoritmus ze skupiny divizních algoritmů, který byl vyvinut pro měření směrodatné odchylky. Divizní algoritmy mají společné to, že počet hledaných klastrů musí jejich uživatel stanovit samostatně.

Hlavní výhoda tohoto řešení tedy spočívá ve správné volbě počtu klastrů, do nichž chceme naše data rozdělit. Existuje heuristická metoda zvaná Elbow method, s jejíž pomocí můžeme zkusit „odhadnout“ jejich optimální počet, ale tato metoda se bohužel zdaleka ne vždy ukazuje jako efektivní. Zvlášť, pokud jsou data hustě zkoncentrována tak, jako je tomu například v mém případě, kdy se výsledek „loketní“ metody rovnal dvěma. Andrew Ng, profesor na Katedře strojového učení Stanfordské univerzity, se domnívá, že neexistuje univerzální způsob jak vybrat správný počet klastrů a tato volba se provádí

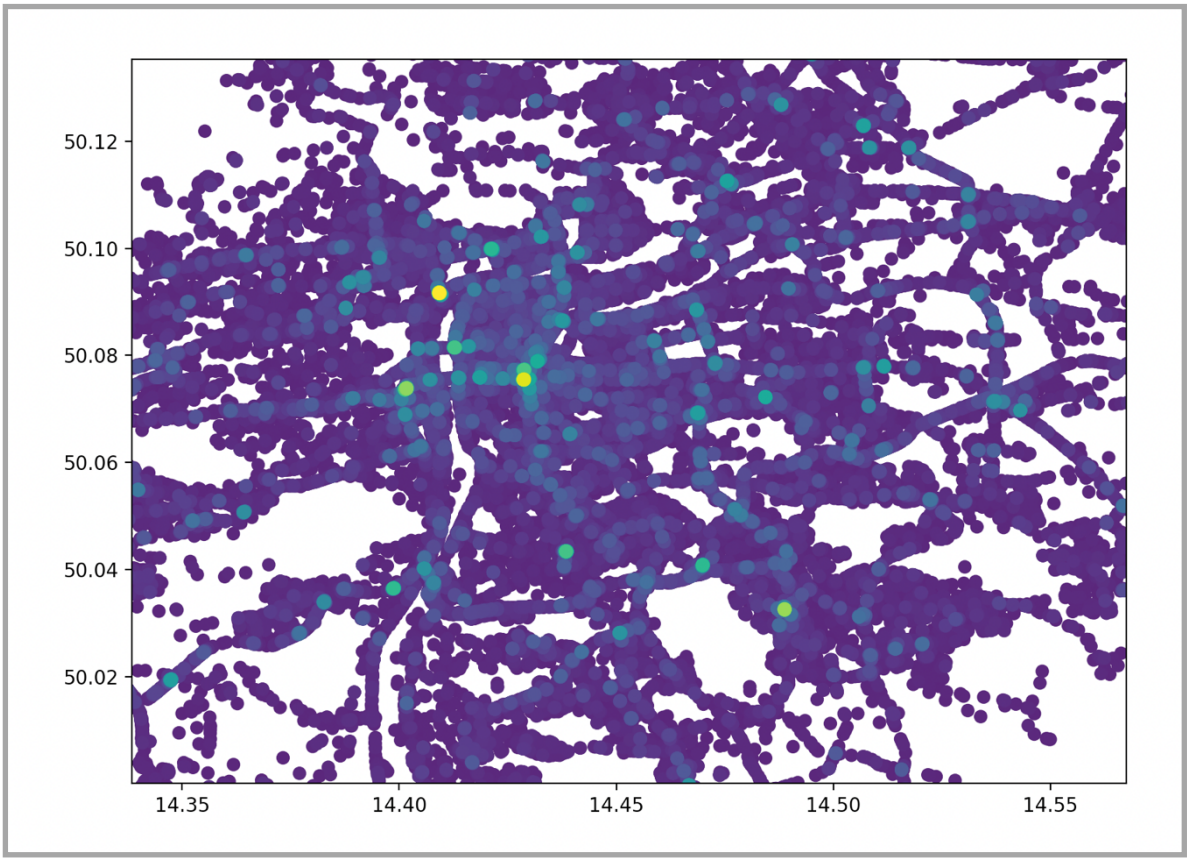
odhadem „podle oka“ (Ng, nedatováno). Realizovav několik pokusů s různým počtem klastrů, rozhodl jsem se pro číslo dvanáct.




*Obrázek 14. The k-means výsledek klasterizace. Zdroj: author.*

### 4.3.2 Dvoudimenzionální histogram hustoty

K vytvoření tohoto diagramu jsem použil kód ze Stack Overflow (Guillaume, 2018). Tento způsob považuji za nejvíce reprezentativní a vhodný. Jeho smysl spočívá ve vytvoření dvoudimenzionálního histogramu, v němž bude s pomocí barvy znázorněna hustota koncentrace bodů. V daném případě platí, že čím více se barva přibližuje k červené, tím vyšší je hustota dopravních nehod v tomto úseku. Tento způsob umožňuje získat vyčerpávající informace o každém konkrétním úseku. Zbývá pak pouze vybrat body, které nás zajímají, najet na ně kurzorem a získat tak jejich zeměpisné souřadnice. Následně již můžeme s pomocí funkce vyhledávání přesně zjistit úsek odpovídající příslušným souřadnicím.




*Obrázek 15. 2D histogram hustoty. Zdroj: author.*

Rád bych ještě jednou zdůraznil, že data, která se mi podařilo shromáždit, jsou z období let 2007 až 2014 a tvoří pouze malou část veškerých dat, která jsou obecně k dispozici. Pro získání aktuálnějších informací by bylo nutné doplnit datovou sadu. Prohlédl jsem si fotografie z míst dopravních nehod, které byly pořízeny v minulosti a srovnal je s aktuálními snímky. V dvou případech můžeme pozorovat provádění změn, které ukazují na existence problém, nepřímo potvrzují získané výsledky.Všechny fotografie níže jsem čerpal z Google Maps (Google, nedatováno).

# 5 Výsledky a diskuse

1. Náměstí I. P. Pavlova na Praze 2 je v souladu s výsledky mé práce nejvíce rizikovým silničními úsekem z hlediska počtu dopravních nehod. Jedná se o významný dopravní uzel s velkým množství křižovatek, kde se křižují tramvajové koleje se silničními komunikacemi a pěšími cestami. Dvě široké dopravní tepny bez jakýchkoliv umělých opatření a bariér, které by mohly bránit nedodržování rychlostních limitů. I přestože je doprava řízena v opačných směrech a rozdělena 2 křižovatkami, zůstává počet dopravních nehod, k nimž zde dochází, mimořádně vysoký.

*Obrázek 16. Náměstí I. P. Pavlova, Praha 2 (Google, nedatováno).*

2. Silničním úsekem, který podle počtu dopravních nehod zaujímá druhé místo hned po Náměstí I. P. Pavlova, je křižovatka mezi ulicemi Radlická a Kartouzská na Praze 5. Opět se jedná o významný dopravní uzel, v jehož blízkosti se pohybuje velké množství aut. Mezi roky 2014 a 2017 bylo na křižovatce použito nové dopravní značení, které zakazuje zastavovat na 2 úsecích. Můžete tohle pozorovat na satelitním snímku.

*Obrázek 17. Křižovatka mezi ulicemi Radlická a Kartouzská, Praha 5 (Google, nedatováno).*

3. Ulice Pod bruskou na Praze 1. V těsné blízkosti se nachází jeden z turisticky zajímavých bodů – Pražský hrad. Zastavují zde turistické autobusy, je tu velké množství turistů. Přes širokou silnici vede přechod pro chodce. Na snímcích z října 2019 lze spatřit umělé bariéry omezující šířku jízdních pruhů, značení rozdělující komunikaci na tramvajové koleje a jízdní pruhy pro automobilovou dopravu, bezpečnostní ostrůvky a nové vodorovné dopravní značení, což svědčí o skutečnosti, že zde byla přijata opatření ke snížení počtu dopravních nehod. Je také možné, že tato opatření byla realizována proto, aby se stal první výjezd určený pro odbočení doleva, následující hned za přechodem pro chodce, bezpečnějším, protože se jedná o velmi nebezpečné místo, kdy řidiči v podmínkách omezeného výhledu křižují čtyři jízdní pruhy.


*Obrázek 18. Pod Bruskou, Praha 1, květen 2009 (Google, nedatováno).*

*Obrázek 19. Pod Bruskou, Praha 1, listopad 2019 (Google, nedatováno).*

4. Významný kruhový objezd v blízkosti obchodního centra Chodov na Praze 11. Je zde pravidelně zaznamenáván zvýšený počet dopravních nehod, nicméně vyšší rizikovost a nebezpečí jsou obecně typické pro kruhové objezdy. Havárie na kruhových objezdech nicméně mají jen zřídka za následek těžká zranění, protože pro vjezd na kruhový objezd je potřeba snížit rychlost. Pokud ke srážce přece jenom dojde, nejčastěji se jedná o boční srážky automobilů jedoucích ve stejném směru (Turnbull, 2004) (The Insurance Institute for Highway Safety, 2020). Existují však i názory, že doprava na kruhových objezdech může být nebezpečnější pro chodce (Turnbull, 2004). Nicméně z důvodu toho, že v případě daného

kruhového objezdu není pohyb chodců vůbec předpokládán, je kruhový objezd zde tím nejlepším možným řešením. Nicméně bych zde určitě doporučil přijmout opatření zaměřená na kontrolu dodržování rychlostního režimu v tomto úseku.

*Obrázek 20. Křižovatka s kruhovým objezdem, Chodov, Praha 11 (Google, nedatováno).*

# 6 Závěr

Prací na projektu, který je podstatou této práce, jsem dokázal nalézt řešení obou úkolů, které jsem si v jeho rámci vytyčil. Za prvé se mi podařilo vytvořit databázi obsahující ohromné množství informací o dopravních nehodách včetně velkého množství podstatných znaků, které jednotlivé havárie popisují. Tato data jsem doplnil o zeměpisné souřadnice každé z nehod. Na základě shromážděných dat jsem provedl analýzu nejnebezpečnějších dopravních úseků z hlediska frekvence evidence dopravních nehod na území České republiky, jmenovitě v Praze a jejím okolí, na něž jsem se musel soustředit z důvodu toho, že webová služba, z níž jsem data po celou dobu práce na projektu shromažďoval, byla vypnuta. Následně jsem vytvořil mapu nejnebezpečnějších oblastí a mnou identifikované rizikové dopravní úseky v dané práci prezentoval.

I když jsem byl bohužel nucen se z důvodu nepředvídatelných okolností omezit z větší části jen na bezprostřední území města Prahy, přesto se domnívám, že obdobné algoritmy mohou být v případě zájmu aplikovány i k jiným obdobně tematicky zaměřeným zdrojům tak, aby celkový obraz byl detailnější.

Tato práce byla pro mě kromě jiného také výbornou praxí a umožnila mi seznámit se s řadou technologií nezbytných k řešení různých úkolů, které v průběhu pracovního procesu vznikají.

Výsledkem práce jsou materiály, na jejichž základě je možné realizovat další výzkum a získávat další a dříve neobjevené znalosti. Také jsem v dané práci uvedl příklady konkrétních silničních úseků, které si zaslouží zvláštní pozornost.

# 7 Seznam použitých zdrojů


World Health Organization. (2018). *extranet*. (WHO) Načteno z World Health Organization: https://extranet.who.int/roadsafety/death-on-the-roads/

Lindberg, H. (04 2017). *Vision Zero 20 years.* Načteno z afconsult: https://www.afconsult.com/contentassets/8f0c19f4f7d24aa5bdbfd338128391ec/2017057-17_0194-rapport-nollvision-eng_lr.pdf

Transport Analysis. (22. 04 2020). *Road traffic injuries.* Načteno z trafa.se: https://www.trafa.se/globalassets/statistik/vagtrafik/vagtrafikskador/2019/vagtrafikskador-2019.pdf?

Python Software Foundation. (2021). *What is python*. Načteno z Python org: https://www.python.org/doc/essays/blurb/

Selenium. (02 2021). *The Selenium project and tools*. Načteno z Selenium dev: https://www.selenium.dev/documentation/en/introduction/the_selenium_project_and_tools/

Google. (nedatováno). *Maps*. Načteno z Google maps: https://www.google.com/maps

Internet Archive. (2021). *Main page*. Načteno z Internet Archive: https://web.archive.org

Richardson, L. (2020). *Beautiful Soup Documentation*. Načteno z Crummy: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Pandas. (nedatováno). *About pandas*. Načteno z Pandas: https://pandas.pydata.org/about/index.html

Ng, E. (nedatováno). *Choosing the Number of Clusters*. Načteno z Coursera: https://www.coursera.org/lecture/machine-learning/choosing-the-number-of-clusters-Ks0E9

Turnbull, G. (2. 10 2004). *Roundabout magic*. Načteno z BBC News: http://news.bbc.co.uk/2/hi/uk_news/magazine/3972979.stm

Zagorujko, N. G. (1999). *Прикладные методы анализа данных и знаний.* Новосибирск: ИМ СО РАН.

Policie ČR. (2020). *Statistické vyhodnocení nehod v mapě*. Načteno z maps.jdvm.cz: http://maps.jdvm.cz/cdv2/apps/nehodyvmape/Search.aspx

Miles, L. (27. 06 2016). *5 ways to calm traffic on a busy road*. Načteno z All traffic solutions: https://www.alltrafficsolutions.com/blog/5-ways-calm-traffic-busy-road/

Straka Jan, P. J. (2020). *ROČENKA NEHODOVOSTI.* Praha: Ředitelství služby dopravní policie Policejního prezidia České republiky.

JGraph. (2020). Načteno z diagrams.net

Johnson, S. (2017). *Want Less Car Accidents? Remove Traffic Signals and Road Signs.* Načteno z Big Think: https://bigthink.com/want-less-car-accidents-get-rid-of-traffic-signals-road-signs

Guillaume. (2018). *How can I make a scatter plot colored by density in matplotlib?* Načteno z Stack Overflow: https://stackoverflow.com/a/53865762

The Insurance Institute for Highway Safety. (2020). *Roundabouts*. Načteno z iihs.org: https://www.iihs.org/topics/roundabouts

# 8 Přílohy

## 8.1 Controller

```
import pickle
import parameters as params
import status_checker
import modifier
import my_browser
import parser
import selenium
import os
from time import sleep

from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
from selenium import webdriver


chrome_options = Options()
chrome_options.add_argument('--no-proxy-server')
chrome_options.add_argument("--proxy-server='direct://'");
chrome_options.add_argument("--proxy-bypass-list=*");
chrome_options.add_argument("--headless")

try:

    browser = webdriver.Chrome()#options=chrome_options)
    browser.get(params.SEARCH_PAGE_URL)
    while True:
        # Get response from status checker
        response = status_checker.check()

        # response == 0 means that there are no date file and it need
to be created
        if response == 0:
            modifier.create_date_file()
        # response == 1 means that we have links to parse information
from
        elif response == 1:
            # read all stuff to a variable
            with open(params.PATH_TO_LINKS, 'rb') as links_file:
                case_numbers = pickle.load(links_file)

            # for every link in file
            for number in case_numbers:
                print(number)
```

```
                sleep(5)
                sources = my_browser.get_sources(browser, number)
                info = parser.get_info(sources)
                modifier.save_info(info)
                modifier.pop_link()

            os.remove('links')
            modifier.change_date()




        # if it's date, it means that there are no links and we need to
get some to work with
        elif response == 2:
            with open(params.PATH_TO_DATE, 'rb') as date_file:
                date = pickle.load(date_file)
            new_links = my_browser.get_new_links(browser, date)
            # Now links are prepared and we need to store it
            modifier.store_links(new_links)

finally:
    browser.quit()
```

## 8.2 Status_checker

```
import os
import parameters as params

def check():
    if (not os.path.isfile(params.PATH_TO_DATE)) or
(os.stat(params.PATH_TO_DATE).st_size == 0):
        return 0

    if (not os.path.isfile(params.PATH_TO_LINKS)) or
(os.stat(params.PATH_TO_LINKS).st_size == 0):
        return 2

    else:
        return 1
```

## 8.3 Browser

```
import parameters as params

from random import randint
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.action_chains import ActionChains
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select
from selenium.common.exceptions import NoSuchElementException
from selenium import webdriver

from time import sleep

def exist_by_xpath(browser):
    """executes to check if dropdown menu exists"""
    try:
        browser.find_element_by_xpath(params.X_DROPDOWN)
    except NoSuchElementException:
        return False
    return True

def collect(browser):
    """executes when dates were filled"""
    sleep(3)
    browser.find_element_by_xpath(params.X_LINKS_OBJ)
    sleep(1)
    case_numbers = []
    links = browser.find_elements_by_class_name('pid')
    for link in links:
        case_numbers.append(link.text)
    sleep(1.5)
    print(case_numbers)
    return case_numbers


def get_new_links(browser, date):
    """executes when file 'links' become empty"""
    # creating browser
    sleep(1.5)

    # find date forms to be filled
    date_from = browser.find_element_by_id(params.ID_DATUM_OD)
    date_to =  browser.find_element_by_id(params.ID_DATUM_DO)

    # fill forms with current processing date
    date_for_form = '.{0}{1}...{2}'.format(date.strftime("%d"),
date.strftime('%m'), date.strftime('%Y'))

    date_from.send_keys(date_for_form)
    date_to.send_keys(date_for_form)

    # click search button
    browser.find_element_by_xpath(params.X_SEARCH_BTN).click()
```

```
    sleep(3.5)

    links = []
    if exist_by_xpath(browser):

        select =
Select(browser.find_element_by_xpath(params.X_DROPDOWN))
        # collect all options
        options = select.options
        for option in range(len(options)):
            select =
Select(browser.find_element_by_xpath(params.X_DROPDOWN))
            select.select_by_index(option)
            sleep(2)
            links += collect(browser)

    else:
        links = collect(browser)

    print('Links collection...OK!')

    return links


def get_sources(browser, link):
    sources = []
    url = params.CASE_URL.format(link,randint(50000,90000))
    browser.get(url)
    sleep(1.5)
    score_page = browser.window_handles[0]
    sleep(1)
    sources.append(browser.page_source)
    # click on button that allow to open google maps by next click on
map
    browser.find_element_by_id("BtnSView").click()

    # select map object
    maps = browser.find_element_by_xpath(params.X_MAP)

    # initialize action chain
    action = ActionChains(browser)

    # add actions to the action chain:  coursor to the map center and
click, then perform action chain
    action.move_to_element_with_offset(maps, params.PAGE_X,
params.PAGE_Y).click().perform()
    sleep(1.5)

    # third window were opened at this moment. Now we need to index it
as prev
    coords_window = browser.window_handles[1]

    # switching focus to the new opened window
    browser.switch_to.window(coords_window)

    sources.append(browser.page_source)
```

```
print('Sources...OK!')

# Clean Up
browser.switch_to.window(score_page)

return sources
```

## 8.4 Parser

```
from bs4 import BeautifulSoup
import re

def get_info(source):
    soup = BeautifulSoup(source[0],'html.parser')
    case_info =''
    case_number = re.search('[0-9]+', ((soup.find("span",
id="LblObjId")).string))
    case_info += ('"' + case_number.group(0) + '"' + '\t')
    for i in ('PnlTabObj1', 'PnlTabObj2', 'PnlTabObj3', 'PnlTabObj4',
'PnlTabObj5'):
        table = (soup.find("div", {"id": i}))
        allinfo = table.find_all('span')[1:]
        for z in allinfo:
            if not z:
                z = "EMPTY CELL!"
            case_info += ('"' + z.string + '"\t')


    soup = BeautifulSoup(source[1],'html.parser')
    coords = re.findall('[0-9]+\.[0-9]+', soup.find("form",
id="form1")['action'])
    case_info += ('"' + coords[0] + '"' + '\t' + '"' + coords[1] + '"'
+ '\n')
    print("CASE INFO " + case_info)

    return case_info
```

## 8.5 Modifier

```
import parameters as params
from datetime import date, timedelta, datetime
import pickle

def create_date_file():
    mydate = datetime(*params.START_DATE)
    with open(params.PATH_TO_DATE,'wb') as date_file:
        pickle.dump(mydate, date_file)
    print('Date creating...OK!')

def pop_link():
    with open(params.PATH_TO_LINKS, 'rb') as links_file:
        links = pickle.load(links_file)
    with open(params.PATH_TO_LINKS, 'wb') as links_file:
        pickle.dump(links[1:], links_file)
    print('Link popping...OK!')

def save_info(info):
    with open(params.PATH_TO_DATABASE, 'a') as database:
        database.write(f'{info}\n')
    print('Colelction...OK!')

def store_links(links):
    with open(params.PATH_TO_LINKS, 'wb') as links_file:
        pickle.dump(links,links_file)
    print('New Links...OK!')

def change_date():

    with open(params.PATH_TO_DATE, 'rb') as date_file:
        file_date = pickle.load(date_file)
        file_date += timedelta(days = 1)

    with open(params.PATH_TO_DATE, 'wb') as date_file:
        pickle.dump(file_date, date_file)

    print('Date change...OK!')
```

## 8.6 Parameters

```
#MAIN

# TELEGRAM CONST

BOT_TOKEN = 'XXX'
BOT_CHAT_ID = 'XXX'

#-----
PREFIX =
'https://api.telegram.org/bot{0}/sendmessage?chat_id={1}&parse_mode=mar
kdown&text='.format(BOT_TOKEN, BOT_CHAT_ID)


# URLS

SEARCH_PAGE_URL =
"http://maps.jdvm.cz/cdv2/apps/nehodyvmape/Search.aspx"

CASE_URL =
'http://maps.jdvm.cz/cdv2/Apps/NehodyVMape/Detail.aspx?objid={}&WinName
={}'

# PATHS

PATH_TO_CONTROLLER = 'controller.py'
PATH_TO_LINKS = 'links'
PATH_TO_DATE = 'date'
PATH_TO_DATABASE = 'database'


#ID'S & XPATHS

ID_DATUM_OD = 'TcoMain_TpaObject_TxtDatumOd'
ID_DATUM_DO = 'TcoMain_TpaObject_TxtDatumDo'

X_SEARCH_BTN = '//*[@id="TcoMain_TpaObject_BtnSearch"]'
X_DROPDOWN = '//*[@id="TcoMain_TpaResObject_LbxResIntervalsObject"]'
X_LINKS_OBJ = '//*[@id="TcoMain_TpaResObject_TblResObject"]'
X_MAP = '//*[@id="OpenLayers_Layer_Vector_82_svgRoot"]'

# DATES

START_DATE = (2007,1,1)


# MISC

WAIT_BETWEEN_RESTART = 600

# COORDS

PAGE_X = 477.5
PAGE_Y = 269.5
```

## 8.7 Main

```
from subprocess import run
from time import sleep
import requests
import parameters as params

def telegram_bot_sendtext(bot_message):
    """Send message to my telegram in case of error"""
    send_text = params.PREFIX + bot_message
    response = requests.get(send_text)
    return response.json()

# Path and name to the script you are trying to start

def start_script():
    try:
        # Make sure 'python' command is available
        run("python " + params.PATH_TO_CONTROLLER, shell = True)
    except:
        telegram_bot_sendtext(An error has occurred!')


start_script()
```